Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.

A COMPARISON OF 2D LINE CLIPPING ALGORITHMS

Václav Skala, Ivana Kolingerová, Petr Bláha
Department of Informatics and Computer Science
University of West Bohemia
Americká 42, 306 14 Plzeň, Czech Republic
e-mail: {skala|kolinger|blaha}@kiv.zcu.cz

Abstract. A comparative study of new line clipping algorithms for 2D convex polygons is presented and the evaluation of algorithms' efficiency based on theoretical analysis and experimental results is given. Complexity of all compared algorithms is O(N).

Key words: line clipping, convex polygon, dual space application, computer graphics.

1. Introduction

Line clipping by convex polygons is one of the most frequent problems in computer graphics. Many algorithms solving this problem have been published so far. The fastest ones are Cyrus-Beck's (CB) algorithm and its published modifications.

Some new modifications were developed and compared with the original Cyrus-Beck's method.

2. Cyrus-Beck's algorithm

This algorithm utilizes the fact that the sign of the dot product of a normal vector of the given edge and a direction vector of the clipped line can give information whether a normal vector of the edge is oriented towards the line segment origin or vice versa. This criterion divides polygon edges into 2 groups with respect to the clipped line; each of these groups offers one intersection point. A slightly shortened algorithm [2] is shown in alg.1.

For all algorithms the following notation is used :

x_A, x_B are endpoints of the clipped line segment,
s is the direction vector of this line, i.e. $s = x_A - x_B$,
N is a number of vertices of the clipping polygon,
x_i, $i = 1, 2, \ldots, N$, are the vertices of the polygon, $x_i = [x_i, y_i]^T$,
$\hat{s}_i$, $i = 1, 2, \ldots, N$ are the direction vectors of edges,
n_i, $i = 1, 2, \ldots, N$ are normal vectors of edges,
t_{min}, t_{max} are the minimal and maximal parameter values on the clipped line or line segment.

Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.

The notation used in this comparative study was chosen as similar as possible to that used in papers describing compared algorithms.

ALGORITHM 1

```
procedure Clip_ 2D_Cyrus Beck (x_A, x_B);
begin
    { n_i is a normal vector, e.g. n_i = [ŝ_y, -ŝ_x]^T }
    { n_i must point out of the convex window }
    {all vectors n_i precomputed; algorithm shortened }
    i := 1; s := x_B - x_A;
    t_min := 0.0; t_max := 1.0; { for clipping a line segment }
    { t_min := -∞; t_max := ∞; for clipping a line }
    while i <= N do
    begin
        ξ := s^T n_i; s_i := x_i - x_A;
        if x <> 0.0 then
        begin
            t := s_i^T n_i/ξ;
            if ξ > 0.0 then t_max := min(t, t_max)
                       else t_min := max(t, t_min)
        end
        else Special case solution;
            { the line is parallel to the i-th edge of the polygon }
        i := i + 1
    end; { while }
    if t_min > t_max then EXIT;
{ < t_min, t_max >= ∅ }      { recompute endpoints of the line segment if changed;
     for clipping a line, endpoints have to be always recomputed }
    if t_max < 1.0 then x_B := x_A + st_max;
    if t_min > 0.0 then x_A := x_A + st_min;
    SHOW_LINE(x_A, x_B);
end; { Clip 2D Cyrus Beck }
```

3. Modified Cyrus-Beck's algorithm

The algorithm proposed in [7] tests first whether the clipped line intersects the given edge. If not, there is no need to compute the intersection point, see fig.1.

A sign of the z-coordinate of the cross product is used as a criterion. The intersection point of the given line with the edge x_0x_1 (with direction vector $\hat{s}_0$) can be detected using condition $[sxs_4]_z * [sxs_1]_z < 0$, similar to [3]. The intersection point for the edge x_4x_0

Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.

need not be computed, as $[sxs_4] * [sxs_0]_z > 0$. The edge which is intersected by the line can be found according to the rule that this edge cannot have both vertices on the same side of the line. This test can easily be done by comparing signs of the clipped line equation in both vertices. The modification of the algorithm from [7] is shown in alg.2. Only the separating function is changed to $F(x,y) = Ax + By + C$ and the vectors s_i are not computed at all. It can be seen that this version of clipping algorithm needs even less computing.

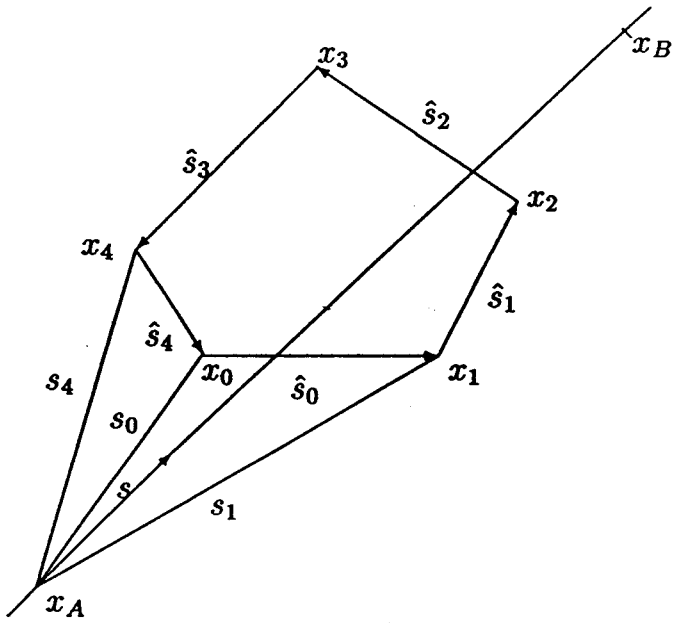


Fig. 1. There is no need to compute the intersection point.

Solution of singular cases can be found in [7].

ALGORITHM 2

procedure Compute(x_A ,x_B);
begin
{ Compute coefficients A, B, C of the line on which segment $x_A x_B$ lies }
end;

procedure Clip_2D (x_A , x_B);
begin {all vectors $\hat{s}_i$ are precomputed}
Compute(x_A,x_B)
$k := 0$; $i := N$; $j := 1$;
$s := x_B - x_A$; $\xi := F(x_i)$;
while $(i <= N)$ **and** $(k < 2)$ **do**

Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.


```
    begin
        η := F(x_j);
        if (ξ * η) < 0.0 then { intersection exists }
        begin
            k := k + 1;
            index_k := i;   { save edge index having intersection }
        end
        else if (ξ * η) = 0.0 then Special cases { ξ = 0 or η = 0 }
                               else Intersection does not exist;
        i := j; j := j + 1; ξ := η
    end; { while }
    if k = 0 then { intersection does not exist } EXIT;
    { k = 2 intersections exist - edges saved in index_k }
    tmin := 0.0; tmax := 1.0; { for clipping a line segment }
    {t_min := −∞; t_max := ∞; for clipping a line }
    i := index_2; t_1 := det[x_i − x_A| − ŝ_i]/det[s|ŝ_i];
    { for effective implementation use identity x_i − x_A = s_i }
    i := index_2; t_2 := det[x_i − x_A| − ŝ_i]/det[s|ŝ_i];
    { recompute endpoints if changed }
    if t_1 < t_2 then { swap t_1 < − > t_2 values ? }
    begin
        if t_2 < t_max then  x_B := x_A + st_2;
        if t_1 > t_min then  x_A := x_A + st_1
    end  { t_2 < t_max, t_1 < t_min can be left out for lines }
    else
    begin
        if t_1 > t_min then  x_B := x_A + st_1;
        if t_2 < t_max then  x_A := x_A + st_2
    end ;
    SHOW-LINE( x_A , x_B );
end ; { Clip_2D }
```

4. Dual space algorithm

The above mentioned principle of comparing signs in vertices can be reversed by dual transformation between points and lines according to [1, 4, 5, 10]. This transformation can be defined for E^2 as follows :

Let's denote

$$p = [p_1, p_2]^T \text{ a point in } E^2$$

Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.

and

$$L(a,c) = \{x \in \mathrm{E}^2 : x^T a = c\},\ a \in \mathrm{E}^2 - \{0\},\quad c \in \mathrm{E}^1,\quad a_2 \neq 0$$

a non-vertical line in E^2. Then the dual image $D(p)$ of the point $p \in \mathrm{E}^2$ is the line. Its equation can be written as

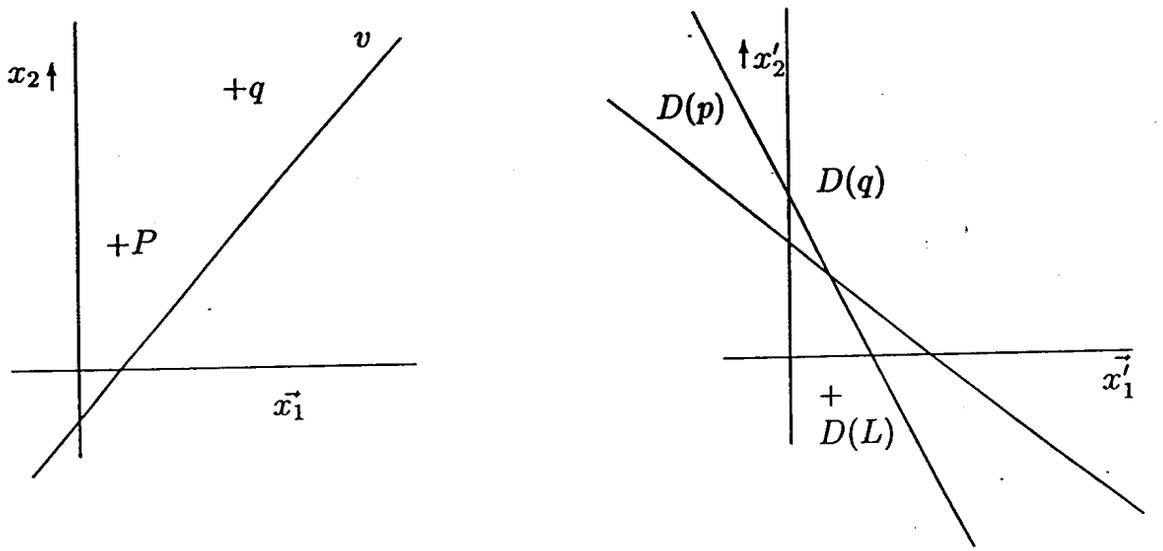
$$x'_2 = -p_1 x'_1 + p_2,$$

that means that the coordinates of a point are used as coefficients in line equation in a dual space representation.

The dual image $D(L)$ of the line L is the point with coordinates

$$x'_1 = -a_1/a_2,$$
$$x'_2 = c/a_2.$$

An example of the dual tranformation can be seen in Fig.2.



$$p = [1,2]^T,\quad q = [3,4]^T,\quad L : x_2 - x_1 = -0.5$$
$$D(p) : x'_2 = -x'_1 + 2, D(q) : x'_2 = -3x'_1 + 4,\quad D(L) = [1,-0.5]^T$$

Fig. 2. Two points and a line segment in the original space E^2 and in the dual space $D(\mathrm{E}^2)$.

The clipped line is transformed into a point and vertices of the polygon into lines. If theirs line equations are evaluated at a point which belongs to a dual image of the clipped line, the intersected edges can be found according to the sign test. This separating function is even more simple than one shown in chap. 3. Nevertheles, we need some extra computing for dual transformation at the beginning of the algorithm.

The shortcoming of this method is that it is not usable for vertical lines (see transformation equations). This case must be solved as a singular case.

The algorithm based on dual representation is shown in alg. 3. (Vertical line case is omitted because singular and special cases are solved in detail in [8].)

This method can be extended for higher dimensions, too, as generally k-dimensional objects in E^n are transformed into $(n-k)$-dimensional objects in $D(E^n)$. In E^3 -case, this principle can be used for polyhedra intersection computation ([8,9], application for ray tracing see [6]).

ALGORITHM 3

```
function SGN(x : real) : integer;
{ Function returns 0,1,-1 according to the sign of x }
begin
    if x > 0 then sgn := 1
    else
        if x < 0 then sgn := -1
                 else sgn := 0
end; { SGN }

procedure Clip_2D_Dual ( x_A , x_B );
begin
    { transformation of clipped line into a dual space }
    s := x_B - x_A; {s = [sx, sy]^T}
    if sx = 0 then Special case solution
    else
        begin
            { computing dual image of line D(L) = [x'_1, x'_2]^T }
            x'_1 := sy/sx; x'_2 := (sx * y_A - sy * x_A)/sx;
        end;
    s1 := SGN(y_N - x_N * x'_1 - x'_2);
    i := N; j := 1; k := 0;
    while (j <= N) and (k < 2) do
    begin
        s2 := SGN(y_j - x_j * x'_1 - x'_2);
        if s1 * s2 <> 1 then
        { the edge i - j is intersected }
        begin
            k := k + 1;
            index_k := i; { save edge index having intersection }
        end;
        s1 := s2; i := j; j := j + 1;
    end; { while }
    { Intersection is computed in the same way as in alg.2 }
end; { Clip 2D Dual }
```

Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.

5. Theoretical comparison and experimental results

If we try to estimate time demands of the proposed algorithms, we count only FPP operations as we consider them to be most substantial for the total time amount. For PC 486/33 MHz we got the following time table :

operation	:=	<	+	*	/
time [0.1s]	33	50	16	20	114

Times shown in the table are for 5.10^6 operations.

For asymptotic cases in algorithms 1-3 it can be derived:

$T_1 = 621 * N$ (Cyrus-Beck),
$T_2 = 175 * N$ (Modified CB),
$T_3 = 205 * N^*$ (Dual space alg.),

* $155 * N$ if SGN is implemented with comparison.

All described algorithms were implemented in C++. Sets with 10^3 line segments each with different percentage of lines intersecting the clipping polygon were used for testing. The endpoints of line segments were randomly and uniformly generated inside a circle. This type of data set was chosen in order to eliminate an influence of rotation. Tested polygons were regular, N-sided and inscribed into a circle (smaller than that used for line segments generation).

The results for 80% of the lines intersecting the clipping polygon are given in tab.1.

N	3	4	5	6	7	8	9	10
ν_{12}	2.33	1.22	1.50	1.64	1.21	1.47	1.30	1.72
ν_{13}	1.11	1.22	1.10	1.49	1.50	1.49	1.48	1.91
ν_{23}	0.48	1.00	0.73	0.91	1.24	1.02	1.13	1.11

N	20	30	40	50	60	70	80	90	100
ν_{12}	2.86	2.67	3.75	2.96	3.03	2.82	2.99	3.15	3.29
ν_{13}	2.68	1.95	2.38	2.27	2.35	2.53	2.35	2.55	2.39
ν_{23}	0.94	0.73	0.64	0.77	0.78	0.90	0.79	0.81	0.73

Tab. 1. Experimental results for PC 486/33 MHz.

Coefficient ν_{ij} in a table is defined as a rate of computing time of *alg.i* to *alg.j*:

$$\nu_{ij} = \frac{T_{alg.i}}{T_{alg.j}}.$$

For $N \to \infty$, the expected values of ν_{ij} are :

$$\nu_{12} = 3.55, \quad \nu_{13} = 3.03, \quad \nu_{23} = 0.85.$$

Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.

These theoretical results are in a good correlation with experimental results.

It can be seen that the efficiency of newly developed algorithms is in all cases higher than the efficiency of CB algorithm. The results of alg.2 seem the best.

It is necessary to point out that the coefficients ν_{ij} do not significantly depend on the percentage of lines intersecting a polygon.

6. Conclusion

Several algorithms for clipping lines by convex polygons were briefly described and compared with each other and with widely known CB algorithm. The results of comparisons show that each of the proposed algorithms should be faster than original Cyrus-Beck's algorithm and can be successfully used for clipping and intersection computing purposes. All tested algorithms have O(N) complexity. A complete list of references on clipping algorithms in E^2 can be found in [11].

References

1979

[1] Brown K. Q. : Geometric Transformations for Fast Geometric Algorithms. PhD. dissertation, Carnegie-Mellon University, Pittsburg.

[2] Cyrus H., Beck J. : Generalized two and three dimensional clipping. C&G, 3(1), 23-28.

1982

[3] Pavlidis T. : Algorithms for Graphics and Image Processing, Springer-Verlag.

1988

[4] Gtnther O. : Efficient Data Structures for Geometric Data Management, Springer-Verlag, Berlin Heidelberg.

1992

[5] Nielsen H. P. : Line clipping using semi-homogeneous coordinates. Submitted for publication in Computer Graphics Forum.

1993

[6] Kolingerová I. : Speeding up ray tracing for convex polyhedra. Proc. Computer Graphics '93, Bratislava.

[7] Skala V. : An efficient algorithm for line clipping by convex polygon. C&G, 17(4), 417-421.

1994

[8] Kolingerová I. : Dual Representation and its Usage in Computer Graphics. PhD thesis, University of West Bohemia (In Czech).

[9] Kolingerová I. : 3D line clipping algorithms - a comparative study. The Visual Computer, 11(2), 96-104.

[10] Nielsen H. P. : An Intersection Test Using Dual Figures, DTU-GK Report 0194.

[11] Skala V. : O (lg N) line clipping algorithm. C&G, 18(4), 517-524.

Machine Graphics&Vision, Vol.3,No.4, pp.625-633, , ISSN 1230-0535, 1995.

Ivana Kolingerová is a lecturer in the Department of Computer Science at University of West Bohemia in Pilsen, Czech Republic. She graduated in 1987 and received PhD in computer science in 1994. Her research interests are computer graphics and its applications. Lately she worked on ray tracing acceleration and, especially, on dual representation and its use in computer graphics.

Petr Bláha was born on 13 June 1970. He graduated in 1993 in computer science at the University of West Bohemia in Pilsen, Czech Republic. Currently he is a PhD student of computer graphics at the UWB specializing in the methods of acceleration of visualisation algorithms. He is the author of two books on object oriented programming.